UNIVERSITY OF GOTHENBURG

Migration of an on-premise application to the Cloud

Master of Science Thesis in Software Engineering and Management

PAVEL RABETSKI

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, October 2011

Migration of an on-premise application to the Cloud

PAVEL RABETSKI

Supervisor: GERARDO SCHNEIDER
Examiner: MIROSLAW STARON

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden

Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden October 2011.

Migration of an on-premise application to the Cloud

Pavel Rabetski

Department of Computer Science and Engineering
Chalmers and Gothenburg University
Gothenburg, Sweden

gusrabpa@student.gu.se

ABSTRACT

Cloud computing has recently became a widely discussed topic in the IT industry. More and more organizations consider using the Cloud, because it enables an easy and cost efficient way of hosting applications, with dynamic scaling and geographical distribution possibilities. Still, it is not clear how and when cloud computing should be used. Existing application are often written in a way that does not really fit a cloud environment well. Also, certain quality attributes (e.g. performance, security or portability) can be affected. More studies are needed on how existing systems should be plugged into the Cloud and what are the consequences of the migration. This thesis aims to share experience and observations we gained from adopting cloud computing for an on-premise enterprise application in a context of a small software company. Our study produced several valuable results. First, main cloud computing opportunities and challenges were identified. Second, biggest cloud platforms were studied and compared. Third, a cloud prototype was developed based on the existing system. Finally, this prototype was used to evaluate the behavior of similar systems in two environments (on-premise and the Cloud) and under different conditions in the Cloud, addressing such concerns as performance and cost.

Keywords: cloud computing, public cloud platform, migration, enterprise application

Acknowledgements

First, I would like to thank my supervisor Gerardo Schneider for his useful guidelines and advice he was giving me through the whole thesis writing. This work would have been impossible without his help.

I am also grateful to Johan Johansson and Mats Svensson from InformaIT who initiated this project and were facilitating it with innovative ideas. They were always open for discussion, providing valuable information.

Finally, I would like to thank all my friends who were assisting me. Their reviews helped to analyze the report from different perspectives.

TABLE OF CONTENTS

1. INTRODUCTION

Cloud computing refers to a utility-based provisioning of virtualized computational resources over the Internet. Even though computing as a utility is not a new term [1], it became commercially available owing to recent technological shifts in virtualization, distributed computing and communication technologies. From a long-held dream cloud computing has turned into a new promising trend of the IT industry that is about to change the way computational resources and software are designed and purchased. Bottery et al [2] believes that the emergence of cloud computing will fundamentally transform the economics of the multi-billion dollar software industry. Market-research firm IDC estimates the market for public cloud products and services growing to $42 billion by 2012 [3], while strategy consulting firm AMI-Partners predicts that small business spending on cloud computing will hit $100 billion by 2014 [4].

Despite such promising predictions, there is a big confusion among potential adopters as cloud computing is not mature enough. Indeed, it is not clear what cloud computing is and when it is useful [5]. According to the Gartner report [6], cloud computing will become the preferred option for application development only around 2015, despite initial growth. Moreover, the lack of standards and keen competition on the new market has led to the variety of idiosyncratic cloud platforms. Cloud giants like Amazon, Google, Microsoft, and SalesForce are trying to establish their rules and promote their franchise. Choosing a proper cloud provider additionally complicates the migration planning, especially for smaller companies that do not have resources for extensive research on cloud computing. This thesis aims to reduce confusion among adopters and provide valuable guidelines regarding migration of existing applications to the Cloud.

The main objective of this work is to analyze what is it to migrate an on-premise application to the Cloud and what are the consequences of the migration. We perform our study on the example of existing enterprise industrial application that is described later in the paper. The main contributions of this work are:

1. A detailed study of the advantages and the disadvantages of cloud computing, and the effects of migration of an on-premise application into the cloud.
2. An evaluation of existing public cloud platforms in order to make a rational choice of a specific one suitable for our purposes.
3. The migration of an industrial enterprise web application to the Cloud.
4. The performance of experiments on the cloud version of our application. Based on our experimental results we draw conclusions on the consequences of the migration and provide suggestions on how to extrapolate our experience to other similar software systems.

The rest of the paper is organized as follows: Section 2 gives necessary background information. Section 3 describes our research methods. Section 4 presents a brief overview of the related work. Section 5 describes the opportunities and the challenges of cloud computing. Section 6 evaluates existing cloud implementations. Section 7 describes the migration of an industrial enterprise system to the chosen cloud provider. Section 8 describes performed experiments and the results. Section 9 summarizes the results and suggests future research direction.

2. BACKGROUND

In this section we give a definition of cloud computing along with its key characteristics. In addition, we describe existing cloud classifications depending on the deployment type and provided capabilities.

Cloud computing

Cloud computing usually refers to a utility-based provisioning of computational resources over the Internet. Widely used analogies to explain cloud computing are electricity and water supply systems. Like the Cloud, they provide centralized resources that are accessible for everyone. Also, in the Cloud you only pay for what you have used. And finally, it is usually consumed by those who have difficulties to produce necessary resources by themselves or just do not want to do that.

Despite the description by analogy, it is difficult to give a unique and precise definition. One of the main ambiguities to define cloud computing is the fact that it is still evolving and taking its shape. The definitions proposed in the cloud computing community are often focused on different perspectives and do not have common baselines. Analyzing existing sources in order to identify common characteristics, Vaquero et al [7] observed no clear and complete definition in the literature. Nevertheless, the authors proposed three features that most closely describe cloud computing: scalability, pay-as-you-go utility model, and virtualization – and gave the following definition:

"Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs."

This definition, similar to other descriptions [8], reveals the main cloud characteristics:

- *Virtualization (abstracted infrastructure)*. Cloud computing became possible through a new evolution of virtualization. Virtualization enables dynamic infrastructure utilization, resource sharing, isolation and security. In contrast to a standard model when processing takes place on specific hardware defined in advance, applications do not have any static computing place in a virtualized cloud environment. Resources are allocated dynamically depending on the demand. Thus, customers do not know the exact place and the type of hardware their applications are running on. Cloud providers can only guarantee minimum performance or storage capacity for the customer.
- *A pay-per-use model*. This is the key characteristic of cloud computing economics. All resources in the Cloud are available on a utility basis, meaning that users are charged based on the quantity consumed by them. This model allows entering the market with no upfront investments into own hardware infrastructure.
- *On-demand access*. On-demand access means that resources like CPU time or storage can be provisioned automatically when needed without any extra management effort.
- *Elastic scalability*. Elastic scaling signifies that computational resources, used by the application, can be dynamically scaled up or down. In other words, virtualized hardware resources can be resized easily and rapidly on demand. It makes a utility model even more attractive, because consumers use only what they really need.

- *Resource pooling*. Computing resources of the provider are shared across multiple users. Different resources are pooled in a multi-tenant way so that they can be dynamically assigned and reassigned to serve consumers' needs.
- *Network access.* Everything in the Cloud is connected via the network. End-users access services via the Internet, developers deploy and monitor applications in the same way, communication between different services in the Cloud occurs through the network. Cloud computing platforms usually provide REST-based APIs to their services.
- *Usability.* Normally cloud computing platforms provide a simple externally managed environment to hide deployment and operating details from the user. Cloud computing systems provide APIs to interact with the environment, which simplifies the development.

Many of these characteristics are well-known from service oriented architecture (SOA), distributed computing, peer-to-peer, etc.

Classifications of the Cloud

There are two widely used cloud computing classifications. The first one describes four cloud types depending on the deployment location:

1. *Public clouds*. Public or external clouds are traditional clouds where resources are dynamically provisioned via the Internet by the off-site third-party providers. These resources are publically available to everyone. Cloud consumers are charged depending on the quantity used. Examples are Microsoft Azure [9], Google App Engine [10], and Amazon Web Services [11].
2. *Private clouds*. Private clouds usually refer to the emulation of a cloud computing environment on private infrastructure. Since users still have to buy hardware and operating equipment, private clouds are often criticized [12][13]. Many companies try this type of cloud to verify their software locally before deploying it to public cloud.
3. *Community clouds*. Community clouds means a cloud environment established across several organizations. Such clouds can be managed by the organizations or third-parties and installed either on- or off-premise.
4. *Hybrid clouds*. This term refers to a composition of two or more clouds, including private clouds and public clouds. This model can be used for different purposes. For example, archiving or replicating local data in the public cloud, or dealing with peak loads when the on-premise system uses the public cloud capacity only when needed.

Another widely used cloud ontology describes three cloud models depending on provided capabilities [14]: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). It is also called a cloud stack (Figure 1) because the cloud models are typically built on top of each other. They can exist independently or in combination with each other. Boundaries between them are still fuzzy due to the lack of standardization.

1. *IaaS.* The cloud infrastructure layer represents fundamental resources that compose the base for upper layers. It is very similar to a regular virtual server hosting. IaaS is built directly on the hardware, providing virtualized resources (e.g. storing and processing capacities) as a service. These resources can be split, dynamically resized and assigned to consumers depending on their demand. In the most common scenario, the consumers are Platform (PaaS) or Application (SaaS) layers that use these resources to build new cloud software environments or applications. IaaS is sometimes subcategorized into computational resources,

data storage, and communication [14]. The examples of public IaaS providers are Amazon Web Services and GoGrid [15].

2. *PaaS.* The platform layer provides a higher level software platform with extended services where other systems can run. This layer is usually built on top of IaaS. It delivers a programming-language-level environment with a set of language-integrated APIs for implementing and deploying SaaS applications, which makes PaaS very comfortable and clear for developers. In addition to that, the platform layer usually has built-in scaling, load balancing, and numerous services for communication, authentication, caching, etc. It further speeds up development, deployment and configuration processes because programmers can focus more on the core logic. Microsoft Azure and Google App Engine are the examples of PaaS.

3. *SaaS.* The services exposed in this layer represent alternatives to locally running applications. They are usually interesting for a wide market, compared to IaaS or PaaS. Services in this layer can be built on top of PaaS or IaaS. They can also be composed from other services available in the Cloud. The application layer provides interfaces for thin or thick clients like web browsers or smartphones. Normally SaaS applications are accessed through web-portals for some fee. Microsoft Office365 or Gmail are examples of cloud services. The SaaS model has proven to be attractive to both providers and consumers [14].

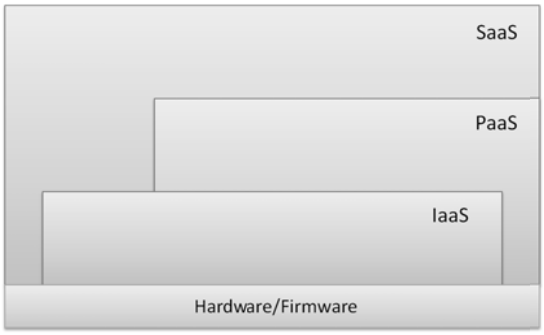

Figure 1 Cloud stack

3. RESEARCH METHOD

We conduct our research using the empirical research method [16]. We are focusing on identifying the required changes in the original application and the consequences of the migration by comparing original on-premise system with new cloud-enabled version. So our research is based on the quantitative approach [17].

First, we studied potential cloud computing advantages and challenges and conduct a comparative analysis of existing cloud platforms, namely, Amazon AWS, Microsoft Azure and Google AppEngine. Later, we apply this knowledge to perform a thorough case study analysis. All data used in the case study is provided by InformaIT and based on one of its products – an on premise enterprise web application. During the case study we identified factors that might affect system behavior in the cloud. We also suggested cloud based architecture for the system and provided a list of required modifications and improvements.

Finally, we performed an experimental research. The main goal of the research was to observe the differences in system performance and cost under variable environment conditions. In order to do the measurements a cloud based prototype of the application was developed. We used the prototype to compare system behavior on-premise and in the cloud, and also against different storage services, deployment location, scale and load in the Cloud.

4. RELATED WORK

Enterprises have to consider the benefits, challenges, and consequences of the cloud adoption when moving to the Cloud. They also need to think over a proper platform for their systems. In this section we present the related work in these areas.

Armbrust et al [5] described they vision of cloud computing, emphasizing elasticity as an important economic benefit. Motahari-Nezhad et al [18] added that significantly reduced upfront commitments and potentially reduced operational and maintenance costs are also important benefits of cloud computing from business prospective. Chappel [19] elaborated on different opportunities that cloud computing brings to ISV, including the potential for more sales and easier customer upgrades. Kim et al [20] made and extensive research on cloud computing issues, emphasizing security and availability as the most challenging ones. Security and privacy seems to be one of the mostly discussed obstacles for cloud computing adoption [21][22].

We have found several papers that evaluated existing cloud implementations. Rimal et al [23] made a comparative technical study of cloud providers and suggested taxonomy for identifying similarities and differences among them. Later, Louridas [24] discussed the migration of applications to the Cloud, examining key features of cloud offerings based on the taxonomy from [23]. Li et al [25][26] suggested a set of metrics related to application performance and cost in a cloud environment, comparing cloud providers based on these metrics. The authors concluded that none of the cloud providers is clearly superior, even though they observed diverse performance and cost across different platforms.

However, we have not observed many publications on the consequences of the migration that would include for example cost, performance, or security comparison. Tran et al [27] provided a simple cost estimation model for cloud applications, based on the identified influential cost factors. Babar et al [28] shared experiences and observations regarding the migration of an existing system to a cloud environment, which also included some guidelines and suggestions. Still, none of the papers compared system behavior before and after the migration (or choosing different migration strategies), like we do in our thesis.

5. ADVANTAGES AND CHALLENGES

Despite numerous advantages, cloud computing brings issues that slow down its adoption. We believe that decision makers need a clear understanding of the advantages and the challenges in order to make a rational decision whether or not to migrate an existing application. In this section we summarize the information collected from different sources [2, 3, 5, 18, 19, 20, 22, 29, 30, 31, 32, 33]. The first part of the section is focused on the cloud computing benefits, and the second part describes its challenges.

5.1. Advantages of cloud computing

1. *No upfront investments*

Companies usually have to create their own data centers to reliably support large software systems. In addition to hardware for computing, networking and storing data, these centers require cooling systems, uninterruptible power supplies (UPS), and other expensive equipment. Hardware and equipment should be purchased and installed in advance to fulfill the expected system load. Furthermore, experienced personnel are needed to manage data centers. Cloud computing eliminates most of these upfront investments. Companies can simply host their applications on cloud providers' data centers, paying only for consumed resources. This model is especially attractive for startups or small organizations because it can significantly reduce time to market.

2. *On-demand capacity*

On-premise installations often struggle with different load patterns. Some patterns can be predicted while others cannot. An example of a predictable pattern is an enterprise service that is under the load only during working hours. A news service, however, meets a heavy traffic after breaking events that are difficult to forecast. A lack of capacity can result in a bad customer experience, while surplus hardware is an inefficient capital allocation. According to statistics, corporate servers are usually more than 80 percent underutilized [5]. Figure 2(a) shows that IT-capacity for on-premise systems is allocated in advance to meet the predicted load. However, the actual load can be lower (which means that hardware is underutilized) or higher (which means that the system is starving). Cloud computing provides much more flexible model of capacity provisioning, when the resources dedicated to the application can be scaled up or down within minutes (see Figure 2(b)). Consequently, the users are able to consume minimum required capacity at any time. This makes a cloud utility model even more financially attractive.

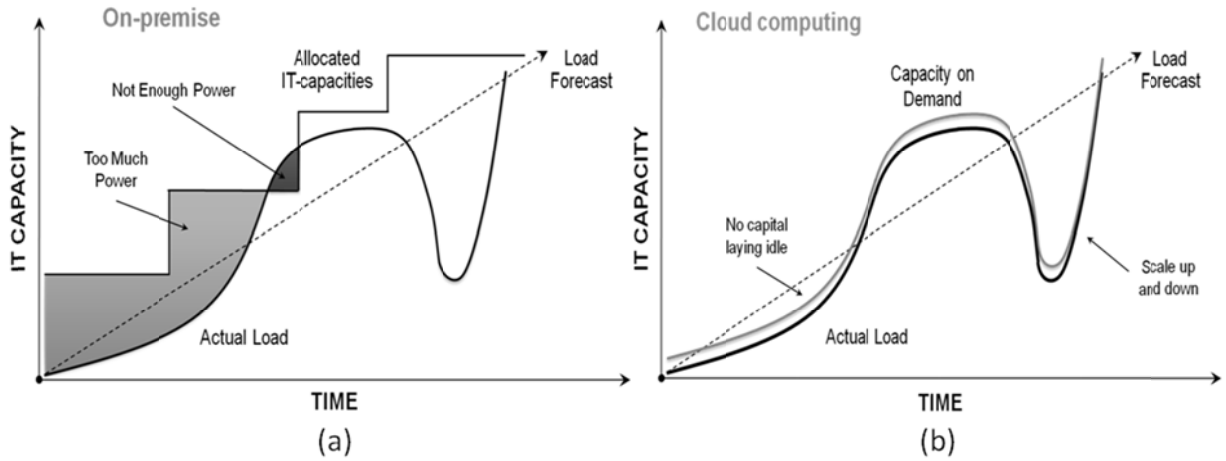

Figure 2 IT-capacity allocation: (a) on-premise, (b) in the Cloud

3. *Focus on core application*

The application stack for standard packaged software consists of many components beyond the application itself, including networking, storage, servers, operating system, middleware, and runtime (see Figure 3(a)). Usually, system providers manage the whole stack, which involves an additional configuration and management effort. Cloud computing reduces this overhead, so companies can be more focused on their primary applications. Figure 3(b) shows the application stack for IaaS. As we mentioned in section 3, IaaS providers take care of networking, storage,

servers, and virtualization, while IaaS users are responsible only for OS, middleware, runtime, data and their applications. PaaS further narrows down vendor's focus to data and applications only. Finally, SaaS completely abstracts end-users from all listed components, providing the necessary service available anytime from everywhere on the Internet.

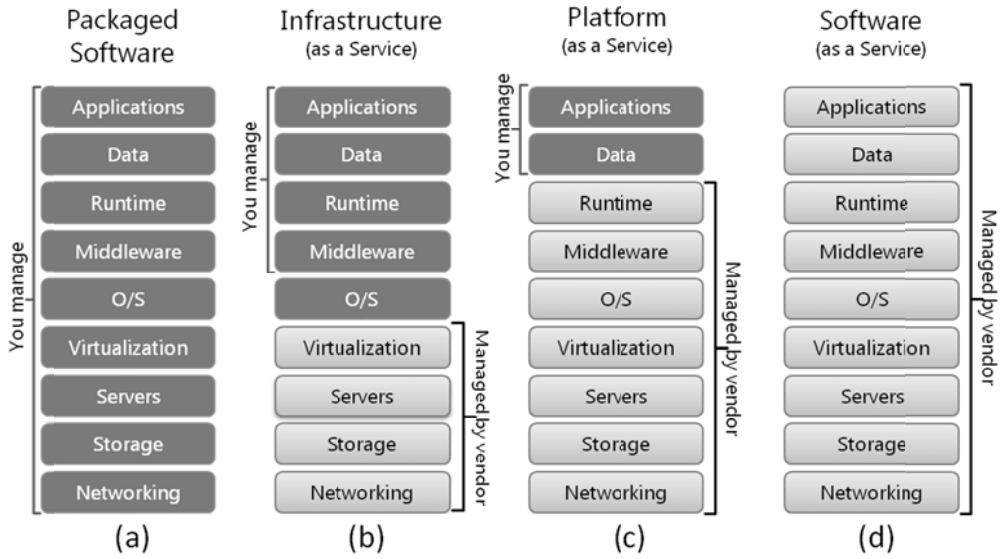

Figure 3 Application stack for: (a) standard packaged software, (b) IaaS, (c) PaaS, (d) SaaS

4. Potential for more sales

SaaS applications are more attractive to end-users compared to regular on-premise systems. Desktop or server installations usually require expensive powerful machines, in-house IT expertise, and technical personnel, while SaaS applications basically need only a web browser. Furthermore, cloud applications do not require any installation or regular updates. Another advantage of SaaS applications is that they usually have a pay-for-use licensing model [5], unlike on-premise commercial products. This model can potentially bring new customers (such as smaller organizations), because they face much lower financial commitments. And finally, the Cloud capacity opens new possibilities for high-performance computing (HPC) applications. Since the use of one CPU during 1000 hours costs the same as the use of 1000 CPUs during one hour, applications can execute hundreds or thousands more tasks in parallel.

5. Easier customer maintenance

Usually on-premise packages are distributed across many customers. Independent software vendors (ISV) have to work with every customer personally, when it comes to installation assistance, upgrading, and handling issues. ISVs often do not have access to a customer environment, which makes it even more difficult. The maintenance overhead grows with the number of users. In contrast to that, cloud platforms allow centralized access to applications. Service providers can easily deploy their applications, delivering updates for all customers simultaneously without any down time. This model also eliminates the need of supporting older product versions. Moreover, developers always have unimpeded access to the necessary information stored in the Cloud.

6. Platform-provided features

Cloud providers apply their knowledge and experience to build their platforms. They use different techniques to address security, availability, and performance issues. For example, high availability

is usually achieved through data redundancy and health monitoring. Cloud providers try to make data replicas independent (including energy, connectivity and hardware independence). So the application keeps running even in case of a natural disaster. Additionally, cloud providers offer geographical data distribution and Content Delivery Network (CDN) services, which decreases latencies and results in a better end-user experience. The same level of global data distribution and redundancy would be very expensive or even impossible to achieve by independent software vendors. There are many more features and integrated services offered in a cost efficient way by public cloud providers, but they should be examined individually.

5.2. Adoption challenges

1. *Security and privacy*

Security and privacy are the most discussed issues of cloud computing. Even though security is improved through data centralization and security-oriented components [34], there is still a concern regarding sensitive information stored in the Cloud. Since users do not fully control their data, they have to trust cloud providers in securing it. Also, the risk of a data leakage on the way to the Cloud brings new challenges regarding secure transportation. VPN or encrypted data tunneling between the local machine and a cloud environment are possible solutions. Private cloud installations were partly motivated by security and privacy concerns.

2. *Availability*

Another cloud adoption issue is availability. Even though cloud providers offer a high level of availability through SLAs, outages do occur in cloud platforms. There are two types of outages: a permanent and a temporary outage. The first one means that the cloud provider goes out of business. A temporary outage means service unavailability during a relatively short period of time like several hours. The biggest cloud providers have experienced several serious outages for the past several years [23]. There are some precautions that cloud consumers can take to mitigate the risk. For example, they can use the Cloud for non-critical systems, keep on-premise backups, and set up a service level agreement. In general, large cloud providers are usually more reliable than small ones.

3. *Performance*

There are also some performance implications when adopting cloud computing. Virtualization and resource sharing lead to performance unpredictability, especially for I/O resources. Cloud platforms should guarantee a fair resource distribution across the applications running on the same machine. Unlike on-premise systems that can keep their code and data in the same runtime environment, cloud components communicate via the network. Since users cannot control the exact deployment location, application components are usually spread across many servers. This results in higher latencies and bandwidth limitations. Performance can become a serious problem, especially when the number of requests and the amount of data increase. Cloud platforms often provide special caching mechanisms and CDN services that can partly compensate these issues.

4. *Compliance requirements*

Many enterprises, especially in the US, are regulated by government policies regarding data security and disclosure, like Sarbanes-Oxley Act for corporate accounting data and Health Insurance Portability and Accountability Act (HIPPA) for people's healthcare insurance data. Most of these rules do not consider cloud services [20], so it is unclear whether or not cloud

computing services violate the regulations. Such issues are not analyzed in our report. However, it should be taken into account when adopting the Cloud.

5. *Vendor lock-in*

Commercial cloud platforms have idiosyncratic implementations which imply different supported programming languages, IDEs, tools, operating systems, integrated services, APIs and unique persistent storages. Cloud platforms have poor interoperability and integration possibilities, so applications become sticky to the provider they are designed for. It makes difficult to design applications that can be easily plugged into several cloud platforms or deployed on-premise and in the Cloud at the same time.

6. *Multi-tenancy*

Traditional on-premise software packages can usually be customized in various ways because they are installed for each customer separately. In contrast to that, SaaS applications are multi-tenant, meaning that a single copy of software is shared by all users. Customization of cloud systems is very limited and requires an extra development effort.

7. *Technological restrictions*

Cloud platforms have different technological restrictions that complicate the application migration. It can be runtime environment restrictions or a limited set of supported languages, frameworks and data storages. For example, .NET applications cannot run on AppEngine, since it supports only Java, Python and Go runtime environments. Also, Microsoft Azure does not support any operating system other than Windows Server 2008. Technological limitations might require a significant or a complete system reimplementation. Legacy systems are usually subject to the risk. Moving these systems to a cloud environment is likely to be expensive due to a large number of required changes.

8. *Licensing*

A regular software licensing model for commercial software does not match cloud computing, because licenses commonly restrict the computers on which the software can run. It brings ambiguities when using supporting commercial software for SaaS applications. Confusing licensing terms and conditions is the biggest obstacle, especially for large organizations considering the Cloud [31]. Even if a cloud-enabling system does not use any supporting software, the system itself might not have a proper licensing model. Software vendors need to reconsider the way they charge for their products in order to sell in the Cloud.

6. PUBLIC CLOUD PLATFORMS

Once ISV has decided to adopt cloud computing (here we consider only the public cloud case), the next step is to choose a suitable cloud platform. In this section we describe three major public cloud platforms, namely, Amazon AWS, Microsoft Azure and Google AppEngine. We emphasize properties that are likely to affect the decision. For example, we describe supported languages and frameworks, runtime environment restrictions, platform-provided services and features, and pricing models.

6.1. Amazon Web Services

Amazon Web Services (AWS) represents a set of online services that form together a cloud computing platform. Amazon has built large-scale, reliable and efficient IT infrastructure where customers can host their applications. Currently Amazon has data centers in five regions: US East (Northern Virginia), US West (Northern California), EU (Ireland), Asia Pacific (Singapore), and Asia Pacific (Tokyo). Besides REST based APIs, AWS has recently released a set of direct language-integrated APIs to access the cloud services.

Compute services

Amazon Elastic Compute Cloud (EC2) service allows renting virtual machines to run custom applications on Amazon's data centers. Virtual machines or "instances" function as virtual private servers. Instances have different CPU resources, available memory, local storage space, and I/O performance, depending on the instance size. The consumers are free to choose any size and deployment region for their virtual machines. In order to instantiate a virtual machine, a user should boot Amazon Machine Image (AMI) that contains operating system with required middleware and configuration settings. It is possible to create custom AMIs or choose available preconfigured images. EC2 is very flexible and supports many operating systems, a lot of middleware, and any development platform or programming framework.

EC2 does not have built-in scaling. The users can manually change the number of instances through administration console or provided APIs. Another possibility is to use Auto Scaling service. Auto Scaling can scale applications up or down dynamically without an extra management effort.

Storage services

AWS offers various durable and scalable storages for different purposes.

Simple Storage Service (S3) provides primary data storage for any type and amount of data. Data is stored in special "buckets" that can be located in a specified region to reduce latencies or cost. Moreover, AWS has a content delivery service for even better data distribution. The provided authentication mechanism allows the protection of sensitive information. Also, S3 has built-in redundancy support, but there is an optional Reduced Redundancy Storage (RRS) service at a lower price.

Amazon SimpleDB is another service used for storing and querying over non-relational semi-structured data. This storage service has a built-in replication, indexing and performance tuning features. Https endpoints ensure a secure, encrypted communication with this service.

Developers can take advantage of *Amazon Relational Database Service* (RDS) to set up and operate a relational database in the Cloud. RDS provides capabilities similar to ordinary databases. In addition to that, it has an automatic replication and backup support. However, developers can still install standard Oracle Database or Microsoft SQL Server on EC2 instances.

Other services

There are many other helpful AWS services for networking, monitoring and controlling, messaging, etc. They can significantly enhance an application development and hosting.

Simple Queue Service (SQS) and *Simple Notification Service* (SNS) are examples of messaging services. They offer reliable communication capabilities among application components and end-users, enabling message-driven and event-driven workflows for large distributed systems. SQS offers a reliable and scalable hosted queue for storing messages that are "polled" by application components. SQS has built-in redundancy support and a special delivery mechanism to achieve high reliability and availability. It fits well to organize communication across EC2 instances. SNS delivers notifications to clients using a "push" mechanism. Potential uses for this service include time-sensitive information updates, applications for monitoring, workflow systems or mobile applications.

Elastic Load Balancer (ELB) is another useful service. It can balance a load for EC2 instances even when the application dynamically scales up or down. ELB can be configured in many ways including sticky load balancing, which means the user is stick to a particular EC2 instance.

CloudWatch is a very powerful monitoring service provided by Amazon. Besides a possibility to track applications, developers and system administrators can configure systems behavior using CloudWatch. For example, applications can be scaled in a scheduled manner or according to certain metrics like CPU utilization. Moreover, CloudWatch can monitor application health and boot new instances in case a failure is detected.

Pricing model and Service Level Agreements

The pricing model for AWS is quite complex. EC2 compute is billed per active instance hours. The price depends on the type and configuration. The users can optionally reserve instances. In this case they get a reduced hourly rate but have to pay in advance. Data storage is charged per GB per month. Data transfer is charged per GB in and GB out. Usually the price is lower within the same region and free within the same Availability Zone. Also, there are additional costs per transaction for some services. Prices vary across different regions.

AWS service level agreements guarantee 99.95% availability of EC2 service, 99.999999999% durability and 99.99% availability of S3 storage, and 99.99% durability and 99.99% availability of RRS. Availability time is calculated for one year period. More detailed information about the pricing model and SLAs is available on the official web site [11].

6.2. Google AppEngine

Google AppEngine is a PaaS offering for developing and hosting web applications on Google-managed infrastructure. One of the biggest advantages of AppEngine is Google's technologies and services available for custom applications. Developers can use standard language-integrated APIs to access most of these services. A set of SDKs and an Eclipse plugin enable full local development support. SDKs can simulate AppEngine environment on a local machine.

Compute services

AppEngine provides a secure environment where applications can be deployed. It currently supports Java, Python and Go runtime environments. Each environment provides standard protocols and common technologies for a web application development. However, regular AppEngine instances have many limitations. For example, access to other computers on the Internet is allowed only through the provided URL fetch and email services; there is a write protection for a local file system; code can be executed only in response to a web request or a task; request has a 30 second limit. In addition to regular instance, developers can use Backends. The *Backend* is an AppEngine instance running in the background. Also, it is more flexible than a regular instance (e.g. it has a higher computational capacity limit and no request deadlines). AppEngine takes care of load balancing and scaling. Applications are scaled based on the load while data is scaled based on the size.

Storage services

AppEngine offers several options to manipulate data. The *Datastore* is used for non-relational data with high read and query performance, auto scaling and transaction support. Unlike relational databases, it supports "schemaless" entities with properties. Datastore offers two types of storage with different availability and consistency.

The Blobstore is another storage service. Developers should use the *Blobstore* for large data objects. These objects stored in Blobstore are called “blobs”. Blobs are usually created by uploading a file through an HTTP request.

Other services

There are other useful services available for developers in AppEngine. *Scheduled Tasks* and *Task Queues* are used to perform tasks outside of the web request. These tasks can be performed according to a configured schedule on a daily or hourly basis; or directly when they are added. The *Memcache* is a cache service that allows building high performance scalable web applications. Memcache represents a distributed in-memory data cache in front of or in place of persistent storage. Additionally, AppEngine has a nice built-in monitoring support. Developers can check collected information for up to 30 days in Admin Console.

Pricing model and Service Level Agreements

Google AppEngine is free for the users up to a certain level of consumed resources. But in general, resources like CPU, storage and bandwidth are billed based on the consumed amount similar to AWS. However, compute services charge per CPU circles but not “per deployment hour”. Since developers do not have a full control over the application scale, Google AppEngine has a preconfigured cost limit of the application. SLA is currently only in a draft version that offers 99.95% availability of custom applications. If Google fails to fulfill SLA, customers receive credits for future AppEngine usage.

6.3. Microsoft Azure

Microsoft Azure is a relatively new PaaS offering in the cloud market. It has data centers in six areas: North and Central America, North and West Europe, East and Southeast Asia. Azure platform consists of Compute, Networking, Storage and Identity services. Figure 5 shows a detailed view of the cloud services within each category.

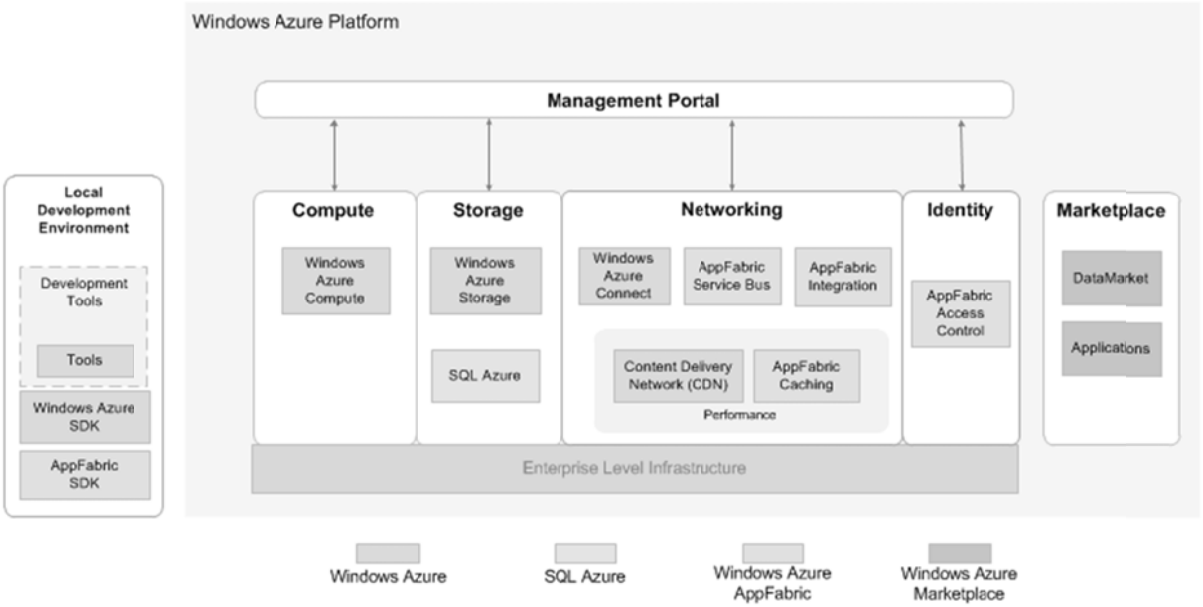

Figure 5 Microsoft Azure Platform products and components

Microsoft Azure provides SDKs and tools for VS2010 to enhance local development. SDKs can emulate a cloud environment on a local machine where developers can run, test and debug applications before moving to the public cloud. Azure extensively uses existing Microsoft tools and technologies like ASP.NET, .NET MVC, ADO.NET, Visual Studio IDE, and Microsoft SQL. So developers can use existing experience to migrate or develop cloud applications for Microsoft Azure.

Compute services

Azure Compute provides a special execution environment for hosted services. Services can be built from different roles. Each role represents a component with unique functionality running inside a virtual server. Azure supports three types of role: Web Role, Worker Role and Virtual Machine Role.

The *Web Role* is intended to run frontend web applications. It has preconfigured Internet Information Services (IIS) 7. IIS7 simplifies the hosting of applications based on web technologies like ASP.NET or WCF. It is also possible to run unmanaged code of virtually all languages including Java or PHP.

The *Worker Role* serves for more general purposes. It is designed to run a variety of code mostly to perform long running tasks or background processing for a Web Role. For example, a Web Role can be used for uploading images while a Worker Role does image processing.

The *Virtual Machine Role* is designed to run user-customized OS images, giving more control over the runtime environment. However, Azure supports only Windows Server 2008 R2 operating system. In contrast to Web and Worker roles that are running inside a virtual machine, VM roles are actually virtual machines. VM Role is useful when moving entire on-premise Windows Server applications to Windows Azure.

Any hosted service is usually represented by a combination of different roles. Roles can communicate with each other directly or by using Storage Services. It is possible to use different capacities for different role instances, since every role runs in its own virtual machine. Table 1 shows five compute instance sizes provided by Microsoft Azure. Each compute instance represents a virtual server with a guaranteed amount of available resources.

Compute Instance Size	CPU	Memory	Instance Storage	I/O Performance
Extra Small	1 GHz	768 MB	20 GB	Low
Small	1.6 GHz	1.75 GB	225 GB	Moderate
Medium	2 x 1.6 GHz	3.5 GB	490 GB	High
Large	4 x 1.6 GHz	7 GB	1,000 GB	High
Extra large	8 x 1.6 GHz	14 GB	2,040 GB	High

Table 1 Microsoft Azure Compute instances

Vertical scaling means changing the size of a compute instance. Horizontal scaling means changing the number of instances. Whereas vertical scaling is limited, horizontal scaling represents the true power of cloud computing. Azure allows dealing with any load pattern in a cost efficient way using dynamic scaling. However, dynamic scaling for roles is not automatic. Service provider has to change the number of role instances manually in the management portal or programmatically by using provided APIs.

Microsoft Azure has a built-in load balancer. It distributes the load across Web Roles to achieve better performance. Furthermore, the platform is constantly monitoring roles to provide a high level of availability. If any instance fails, a new one is reinitialized automatically. Moreover, applications have no down time during upgrades. Microsoft suggests having at least two instances of each role to achieve offered availability.

Storage services

Azure Storage provides scalable, persistent, durable storage in the Cloud. It is exposed as a REST-based service. Data stored in Azure Storage can be accessed from Azure Compute or from anywhere on the Internet through HTTP. The users can configure access policies using built-in authentication. There are four storage abstractions supported by Azure Storage: Blob Storage, Table Storage, Queue Storage, and Windows Azure Drive.

The *Blob storage* is unstructured storage that resembles a regular file system. It provides an interface for storing any named file along with its metadata. It supports a hierarchical structure, similar to a file system. File size as well as the number of files are not limited.

The *Azure Drive* represents a durable NTFS-formatted virtual hard drive (VHD) that uses Blobs as underlying storages, where data is located persistently. This storage can significantly simplify the migration of existing applications, because it allows simple NTFS API calls from the code. However, Azure Drive can be mounted by only one role at a time. Furthermore, it cannot be accessed from outside of Azure environment, meaning that users cannot access its data directly over HTTP.

The *Table Storage* is used for structured or semi-structured data. It provides efficient mechanisms for storing and querying over this data. However, Table Storage should not be mixed up with a relational database. In a simple way, a table is a set of entities with properties. Entities are similar to table rows, while properties are similar to table columns. Entities can have different properties within one table because it has no defined schema. All tables in the storage have mandatory PartitionKey and RowKey properties.

The *Queue Storage* is a special storage that is used to organize an asynchronous, loose coupled workflow among roles. Roles can exchange text messages that are stored in Queues. One Queue can be used by several roles and vice versa. Queue Storage also provides a special delivery mechanism to guarantee high reliability. If any role fails to process a message, the message is recreated in the Queue.

Azure Storage scalability model allows manipulating huge data amounts with fast concurrent access. Frequently used data is cached in memory to meet dense traffic needs. Moreover, data is partitioned across many physical nodes to distribute the load. Tables are partitioned based on the PartitionKey property, and Blobs are partitioned according to their hierarchical structure.

The *SQL Azure* represents a robust relational database provided as a cloud service. It has rich management, monitoring and reporting capabilities. SQL Azure Databases are automatically partitioned across many nodes to distribute the load. Furthermore, SQL Azure supports bi-directional synchronization and data sharing among multiple cloud and on-premise databases. From the development perspective, SQL Azure is almost identical to a regular Microsoft SQL Server.

All data in Azure storages is replicated three times to achieve high availability and fault tolerance. Data replicas are distributed across different fault domains. If one of the replicas breaks, it is automatically recreated from a healthy one. Windows Azure also takes care of data consistency across replicas.

Other services

Most of the rest services provided by Microsoft Azure enhance network-related performance of cloud applications or simplify the migration of existing on-premise solutions to the platform.

The *AppFabric Cache* represents a distributed in-memory cache for Windows Azure applications. The Cache size can be dynamically configured on demand. Furthermore, the same cache model can be used for both on-premises and cloud applications. AppFabric Caching can dramatically reduce application latencies. It can be used as a session state provider or a regular storage cache layer.

The *Content Delivery Network* (CDN) service allows distributing Blob or local content across many more locations, providing minimum latencies and maximum data scale. CDN can significantly improve application throughput. However, only public content can be used in CDN. Also, it is highly discommended for volatile or dynamic data.

The *Access Control* service enables an easy authentication mechanism. It supports standard identity providers including Active Directory, Window Live ID, Facebook, etc.

Pricing model and Service Level Agreements

The pricing model of Microsoft Azure is similar to other cloud offerings. The users pay only for consumed resources without any upfront investments. Though, different subscriptions are possible. Compute services are charged according to the VM instance size on hourly basis. Compute hours is the amount of clock hours the application is deployed (regardless CPU utilization). Both staging and production deployments are counted. Storage services are charged per GB/month and a number of transactions. Storage capacity is calculated as average space used by Blob, Table, Queue, and Driver storages during the billing period. That means 30 GB used only for one day is billed as 1GB/month. More information with the detailed description of standard rates is available in Appendix A.

Microsoft offers a set of SLAs for services including Windows Azure Compute, Windows Azure Storage, Windows Azure CDN, etc. Most service level agreements guaranty minimum service availability and, in some cases, performance. Availability of compute services is 99.95%, of storage services – 99.9%. If these rules are violated, customers receive service credits in compensation.

6.4. Summary

As we observed in this section, cloud platforms are unique in many ways. They not only represent different layers of the cloud stack, but also have specific services and provided features. Consequently, cloud platforms are suitable for performing the migration of existing applications in different ways.

AWS is similar to a virtual private hosting where users can control almost the entire software stack. It gives great flexibility for developers, but makes it difficult to offer automatic scalability, load balancing and failover [5]. Nevertheless, AWS has a variety of integrated services for that. A web service interface makes Amazon's offering really platform and language independent. With its level of flexibility and interoperability, AWS is suitable for the majority of existing applications.

Google AppEngine offers a high level domain-specific platform, targeting traditional web applications. AppEngine looks much like a cloud application framework. It allows automatic load balancing, elasticity and integration with other Google services, but puts applications into a very limited environment. AppEngine is a good choice for building new applications, but the migration of existing systems is likely to require significant reimplementations.

Microsoft Azure intermediates between AppEngine and AWS. It resembles a lower lever application framework, providing a language-independent managed runtime with certain

possibilities to control the environment. Weak points of Azure are the lack of monitoring capabilities and no built-in scaling support. Thus, the users have to purchase external tool/services or develop their own. In general, Azure fits well for the systems based on Microsoft technologies and tools. A consistent development experience is an additional advantage in this case.

All cloud providers have well-developed core services for computing and storage. They offer highly scalable and available persistent data storages where data is automatically replicated and partitioned. Additionally, the platforms have special messaging services that help to organize asynchronous lose-coupled workflow. However, current service level agreements are very weak. In most cases they offer only availability of the service. None of the providers compensates business losses if the agreement is breached. AppEngine has only a draft version of SLA.

Cloud-enabled systems are likely to have similar cost and performance across the platforms [25][26]. However, there are some distinctions they might have. For example, Microsoft Azure tends to be slightly more expensive, but it shows considerably better I/O performance. Also, the scaling speed differs across platforms. The average time to provision an additional instance is about ten minutes in Azure, and only about one minute in AWS. However, it is highly dependent on the configuration (e.g. Windows OS boots much slower than Linux OS).

Also, we have noticed a rapid evolution of the platforms. They are expanding their data centers and developing new services for cloud consumers. One year ago Amazon had only three data center locations (two in US and one in Europe), compared to five locations now. Moreover, cloud providers are competing in pricing. For example, Microsoft Azure has significantly reduced the cost of SQL Azure and stopped charging for incoming traffic.

7. CASE STUDY: MIGRATING DC SYSTEM TO THE CLOUD

In this section we describe the migration of an enterprise system to the Cloud. We follow the migration process provided in [27], presenting the most relevant information here. First, we describe the current system implementation with a short background about the company. Then, we describe the new cloud DC architecture along with identified compatibility issues. We also suggest several system improvements to further leverage the cloud environment. And finally, we outline concerns about the new architecture.

7.1. Current DC implementation

InformaIT company

InformaIT is a small ISV that focuses on document management systems. Most of the systems are based on Microsoft technologies, so developers have rich experience in .NET framework, Visual Studio development environment, SQL Server database system, and Windows Server OS. Being an innovative company, InformaIT is very interested in modern technology and IT trends. Cloud computing is also in the area of interest.

The Document Comparison system (DC) was selected as a candidate for migration. DC is a small web-based enterprise solution that enhances document management processes. The main purpose is to provide a fast and easy way to compare textual and graphical content of different digital documents. The following section gives a detailed description of the system and its architecture.

DC architecture

The system is implemented using Microsoft .NET 2.0 framework and various programming languages including server-side C# and C++, and client-side JavaScript. DC contains five main components: frontend web application, backend engine, distributed cache, database, and shared file store (see Figure 6).

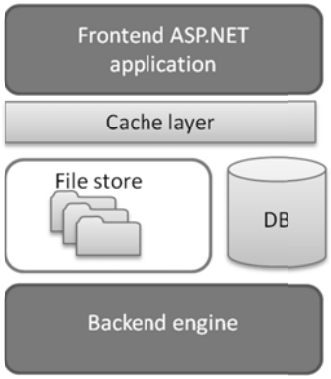

Figure 6 DC components

The *frontend* is a simple ASP.NET web application running under IIS on Windows OS. It provides web interfaces for end-users, so they can upload digital documents, change configuration settings, analyze the result, and generate reports. The frontend extensively uses ASP.NET session state to track processed information and a current user status. The client-side JavaScript reduces the load on the server and improves usability.

The *backend* is implemented as a Windows service (also.NET based). It performs long running computational tasks e.g. the rasterization of digital documents. A special commercial library is used to fasten this process. The library accesses the files via regular file system API. It also requires the registration of a COM component.

The *file store* serves as a shared storage for system components. It keeps persistent data and organizes asynchronous communication between the frontend and the backend. The frontend stores uploaded documents and creates XML task files there. An XML task file describes a job unit for the backend. The backend is checking for new XML task files, and then stores rendered images in the corresponding location. The file system also contains some custom configuration files that are shared among all users of the installation.

The *cache layer* keeps frequently used data, which increases the performance of the system. For example, the frontend stores user session state there. A local in-memory cache is used whenever possible. However, some installations require a distributed cache shared across several web applications.

The *database* contains user personal information. This information is needed for authorization and authentication. The frontend uses the database mostly during login and logout procedures. Unlike many document management systems, DC is not database-centric. The amount of data is quite small and this data is used infrequently.

Deployment model

DC is deployed on servers located in the data centers of customer organizations. This means customers have to take care of the infrastructure and technical personnel to maintain it. The amount of required hardware depends on the amount and the complexity of processing data. A single server is usually enough for small companies, while big organizations need several machines to run the system. DC also requires preinstalled Windows Server 2003/2008 with Microsoft SQL Server.

An on-premise distributed deployment model of DC is shown in Figure 7. ASP.NET applications are composed into a Web Server Farm. They store frequently used data in a distributed cache that is usually located on a separate server. Backend engines are deployed separately as well. They require more powerful servers for heavy computations. A customer can choose the number of frontend and backend servers to achieve the required performance. A shared network folder plays the role of persistent file storage. Microsoft SQL Server is used as a database. Since it does not require much space and heavy querying, the database can be installed on a shared server. Usually end-users are also located in the same environment where the system is running

This on-premise deployment model gives several advantages. First, it keeps data and code physically close. It results in very low latencies and no bandwidth limitations. Second, sensitive data never goes outside the organization, which provides a high level of security. In some cases when users need to access the system outside the organization, a VPN connection is established to keep the transferred data protected.

Organizations are charged per installation depending on the number of users. There are different types of licenses available, including a personal license and a concurrent license.

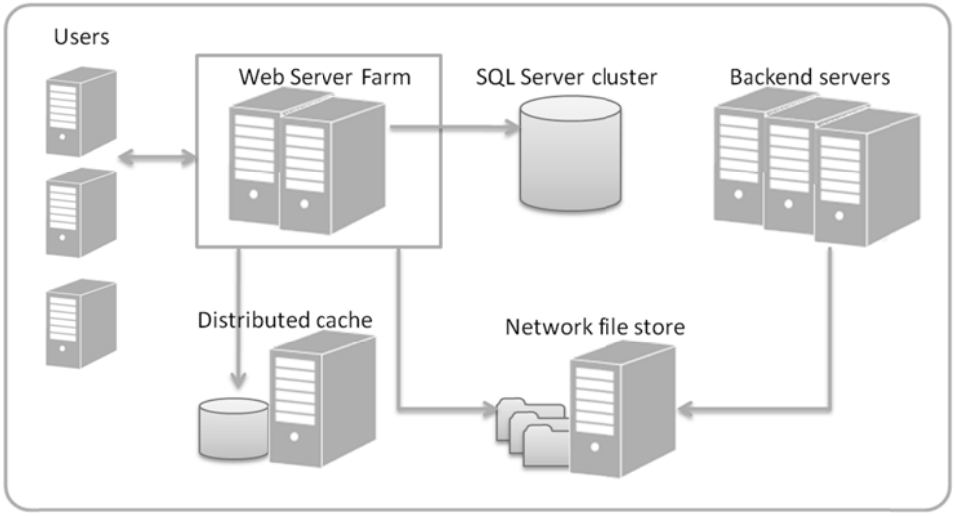

Figure 7 On-premise distributed deployment model of DC

Motivation

InformaIT believes that adoption of cloud computing will solve or mitigate many of the currently faced challenges. The main triggers are the potential for more sales and easier customer maintenance.

On-premise DC is oriented to big and medium organizations that have enough resources, own infrastructure, and technical personnel to install and run the system. Furthermore, the license cost is quite high. Potential customers such as small companies cannot afford DC, facing too big financial commitments. Some of them would like to use the system inconstantly and pay only for the amount of compared data. SaaS version of DC can bring the product to such customers.

Another opportunity is easer installation and upgrade procedures. The system is distributed across many customers. InformaIT has to convince each customer to replace an on-premises package and then assist during the actual upgrading. Some customers still run older versions of the system, which brings an additional support overhead. Simple maintenance model of cloud computing will help to distribute resources more efficiently, leading to cost savings and business agility.

Also, InformaIT rents several virtual private servers for demo installations. These servers cannot be scaled dynamically, remaining underutilized most of the time. Moreover, the deployment location is static, so remote customers experience very high latencies when trying the application. The system can leverage technological edge of the Cloud, including dynamic scalability, geographical distribution, and data replication. Being a small company, InformaIT cannot achieve the same global scale and global reach by expanding own infrastructure. Cloud computing enables these possibilities in a cost efficient way with no upfront commitments.

For now, a cloud version of DC will be used to replace the demo installations. This will help to examine systems' behavior in the Cloud. Meanwhile, InformaIT would like to keep an on-premise DC version. Ideally, the system should be easily deployed in both environments with only few changes.

7.2. Suggested cloud DC architecture

There are many ways to migrate the system to the cloud environment. Developers usually face a range of alternatives when implementing cloud-based systems. In this section we describe the chosen approach for our case and discuss different alternatives that can affect cost, architectural qualities, and the amount of required changes.

Choosing a cloud provider

The first step when moving the system to the Cloud is to choose a proper cloud provider. A properly chosen cloud provider can significantly decrease the effort and the cost of the migration. We have already examined three major cloud providers in section 5. Based on our finding we can conclude that Google AppEngine is the worst candidate for DC because it does not support .NET applications, while Amazon AWS and Microsoft Azure both fit for the migration quite well. After further analysis we prefer Windows Azure to Amazon AWS for several reasons: it requires less configuration effort, has a faster deployment model, and allows consistent development experience for applications that are well-versed in Microsoft technologies.

Cloud DC architecture

Once we have chosen a public cloud provider, we need to show how existing architectural components are mapped to abstractions provided by the platform. In our case this platform is Microsoft Azure.

1. *The frontend*. Azure Web Role is an obvious choice for our ASP.NET frontend. Web Role has a preconfigured IIS and a built-in load balancer for web applications. Still, there are some limitations to keep in mind. For example, Azure load balancer is not sticky, meaning that two requests from the same user can be processed by different Web Role instances. Also, Web Role supports only IIS 7.0 and requires .NET 3.5/4.0.

2. *The backend engine*. The backend maps to a Worker Role, since it suits perfectly for long running background tasks. It is worth noting that roles do not have administrative privileges in the environment. It restricts the execution of tasks that change OS configuration e.g. registration of a COM component or changing OS registry.

3. *The distributed cache*. Microsoft Azure has only one service for distributed cache so far – AppFabric Cache. Alternatively, cached data can be stored in either SQL Azure or regular Azure Storage. As we have explained in section 5, AppFabric Cache is considered to have better performance compared to the alternatives. However, it is quite expensive and limited in size (4GB maximum). We choose AppFabric Cache under the assumption that the size of data stored in cache will be significantly reduced. Otherwise we suggest using Table Storage.

4. *The database*. On-premise DC version uses a Microsoft SQL Server database. We find SQL Azure as a perfect cloud alternative. In most cases switching to SQL Azure is as simple as changing the connection string. In [35] authors argue that SQL Azure can become a bottleneck for systems that concurrently operate large amounts of data. However, it is not the case for DC.

5. *The file store*. We have found out that the local file storage is not persistent and cannot be shared with other roles. All data stored locally gets lost if the role dies. The only persistent option for Azure applications is Azure Storage. We suggest using Queue Storage messages instead of XML task files and Blob Storage for the rest of the files shared among roles. This approach leverages the cloud platform as much as possible. First, all data is automatically replicated and scaled. Second, Azure Storage can be accessed directly via REST calls, reducing the load on the frontend. Last but not least, Queue Storage provides a built-in reliable communication mechanism.

   There is another alternative for the file store: Azure Drive that represents a VHD located in Blob Storage. This persistent drive can actually be shared with other roles using a Server Message Block protocol (SMB) [36]. The advantage of using Azure Drive is that it supports a regular file system API, eliminating any code modifications. However, the drive becomes unavailable if the role that mounted this drive dies. Also, it can store maximum 1TB of data, limiting scalability of the system. Moreover, stored data cannot be reached from outside via REST API.

We prefer the first option because it has easier deployment procedure, can scale without complexity, and provides higher availability.

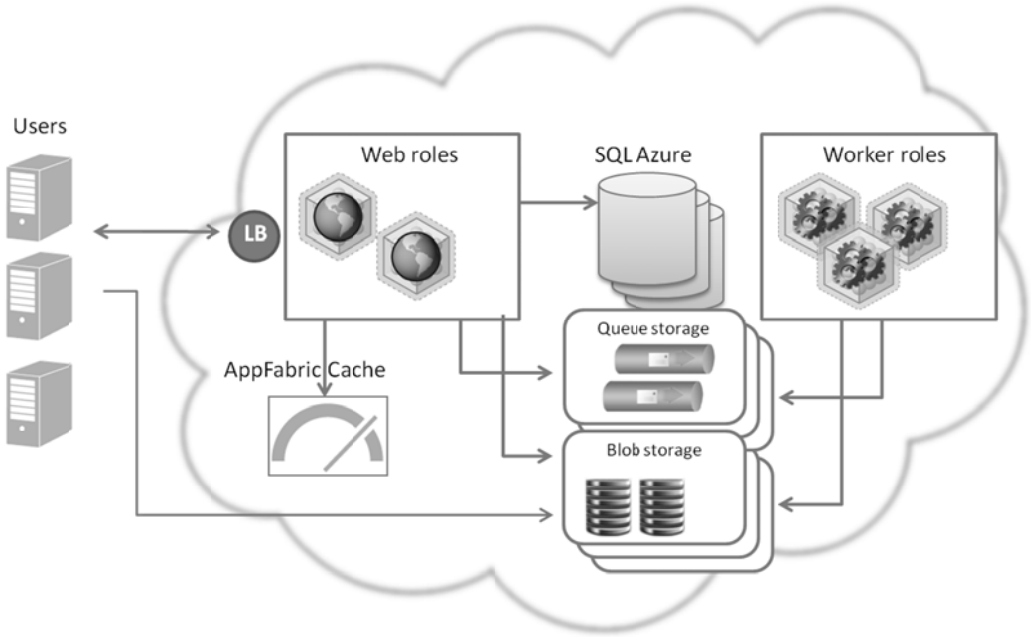

Figure 8 Cloud deployment model of DC

Figure 8 presents a proposed deployment model of the system in the Cloud. DC offloads the complexity to Azure services, taking advantage of built-in efficient data partitioning and replication, global scale and global reach, application's health monitoring and load balancing.

Identified compatibility issues

Even though Microsoft Azure fits well for the migration of DC, we have identified some compatibility issues that require changes in the current implementation. These issues are described in Table 2.

Compatibility issue	Required modification
Current solution uses .NET 2.0 and VS2005 that are not supported by Microsoft Azure. The platform uses the latest product versions.	The system should be migrated to .NET 3.5/4.0 and VS2010. This modification is quite simple due to full backwards compatibility of .NET 4.0 and 2.0.
The system cannot register COM components directly from code due to environment limitations.	There are some workarounds that allow using COM components for Azure applications: Registration-Free COM [37] and role startup scripts. We suggest using startup scripts because it is the easiest solution.
A local folder cannot be shared across Azure roles. Furthermore, suggested Blob Storage and Queue Storage have APIs that are not compatible with regular file APIs currently used by DC.	Change the code for accessing data in the file storage to use Blob Storage and Queue Storage APIs. Since commercial libraries cannot be changed, required files should anyway be stored locally every time before processing. Azure Drive

	is an alternative solution that eliminates these changes.
Standard ASP.NET session state modes do not suit Azure environment. Even though In-Process mode (local in-memory session state storage) is an option for one Web Role instance, it is useless for several roles because of a non-sticky load balancer.	The system needs distributed session storage in order to scale. We suggest using AppFabric Cache (or optionally Table Storage). Microsoft Azure offers an easy way of using these storages as custom session storage providers.

Table 2 Identified compatibility issues

In what follows, we recommend some design modifications in order to tune system performance, increase portability, and make the migration as smooth as possible.

1. Separate data layer from business logic layer

As stated earlier, InformaIT wants the system to by easily portable across both environments. However, this is not easy to achieve because of the need to switch from regular file system API to Azure Storage API. We suggest separating a data access layer from a business logic layer in order to increase portability. In other words, instead of using APIs directly, a business logic layer calls a data access layer interface. This loose coupling allows using regular file system or Azure Storage depending on the deployment environment (see Figure 9).

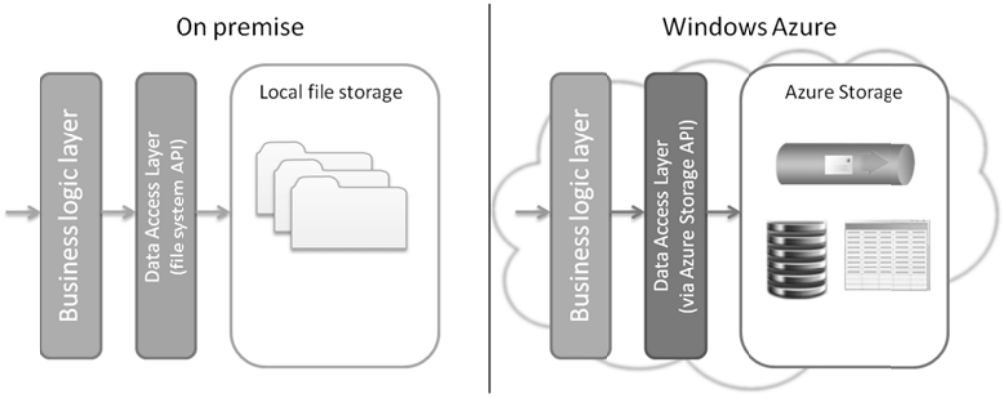

Figure 9 DC design to increase portability

2. Become as stateless as possible

Large amount of cached data will not only degrade the performance but also increase the cost. An additional 1GB of AppFabric cache costs around 100$, which is 1000 times more than Azure Storage cost. The bigger the session size, the more time required to serialize/de-serialize it. DC currently stores megabytes of data per a session, which is a big overhead. We suggest reducing the amount of cached data, making DC as stateless as possible. This suggestion can be applied for any web application that extensively uses session data. Abuse of session is a bad design for cloud applications.

3. Extensively use logging

Logging is very important for cloud applications, since debugging is impossible in the cloud environment. Logging helps developers to trace the behavior of the system and determine the

reason of system failures. Furthermore it might be useful for identifying the level of resource utilization or just collecting statistical information.

Concerns

InformaIT still has some concerns regarding new cloud architecture.

First, there is a performance concern. Unlike an on-premise deployment, a cloud environment entails higher latencies, because all components communicate over HTTP. This might be particularly harmful when frequently storing and retrieving session data. Another potential bottleneck is CPU performance. Heavy algorithms used in DC system consume a lot of CPU resources. We need to examine the application's behavior in the cloud in order to make a final decision upon the feasibility of cloud adoption.

Another concern is the cost of the application, because it is not really transparent in the Cloud. The price model is based on different metrics that are difficult to measure for all system components in common. InformaIT would like to have at least approximate cost estimation for DC running on Microsoft Azure platform.

It would also be interesting to see how a deployment location can affect latencies and bandwidth, because InformaIT is interested to leverage this cloud platform advantage. In the next section we investigate these concerns by doing performance experiments and cost estimations.

8. EXPERIMENTS

In this section we investigate the main concerns of running DC in the cloud environment. These concerns are cost and performance. Based on our estimations and experiments, we compare DC behavior in two environments (on-premise and the Cloud) and under different conditions in the Cloud. This helps us to determine whether cloud-based DC is feasible, which is the main goal of the investigation. Other potential concerns, such as security and privacy, fall outside the scope of this thesis and remain as future work.

Testing environment

Experiments and measurements are done for North Europe deployment location of Microsoft Azure. This is the geographically closest location to the client testing environment located in Sweden (Gothenburg). All Azure compute instances have a small size, which provides 1.75 GB memory, 225 GB local disk space, moderate I/O performance, and CPU performance equivalent to one 1.6GHz core. We use small instances as a part of Azure free trial subscription, which gives necessary resources to perform our experiments for no fees. Testing on the client side is executed in a non-virtualized environment, external to the Cloud, with a direct connection to the Internet via a high-speed wired Ethernet. However, the cloud deployment location and the client environment are changed for some experiments. All experiments are performed at least 100 times to confirm that the results are stable.

8.1. Performance

As we identified earlier, a cloud environment entails increased latencies and unknown hardware underneath. Therefore, DC can have the following performance bottlenecks in the Cloud: the execution of heavy computational tasks (like digital document rendering) that require efficient hardware; and session handling that is latency sensitive. These operations represent the highest risk when moving DC to the cloud environment, because they might lead to significant system performance degradation.

8.1.1. Page rendering time

We use "seconds per page" metric for measuring page rendering speed. This is a natural metric that represents the time required to create an image from a digital document page. To suggest a page standard, we have analyzed a production set of documents and classified two main types:

1. Textual documents with little graphic. They usually contain many (up to 40) A4 format pages that are rendered fast. We refer to this type as A4 page.
2. Graphical documents with little text. They usually contain one or two A3-A2 format pages that are rendered considerably slower. We refer to this type as A3 page.

We pick up one A3 page and one A4 page for our experiments. The size of the A3 page is 25Kb, with a 495Kb corresponding rendered image. The size of A4 page is 2Mb, with a 1.4Mb corresponding image.

The rendering library gets regular file system paths as input parameters. Since the original on-premise system keeps all documents in a regular file system, it calls this library directly. However, as we described in section 6.2, cloud DC uses Azure Storage. This requires some pre-processing and post-processing steps to render a page:

1. Download a required document from Blob Storage to local file storage
2. Call the rendering library
3. Upload rendered images to Blob Storage
4. Cleanup local storage

So the total rendering time in the Cloud can be decomposed into these four steps. Figure 11 illustrates the observed time for A3 and A4 pages in the cloud environment.

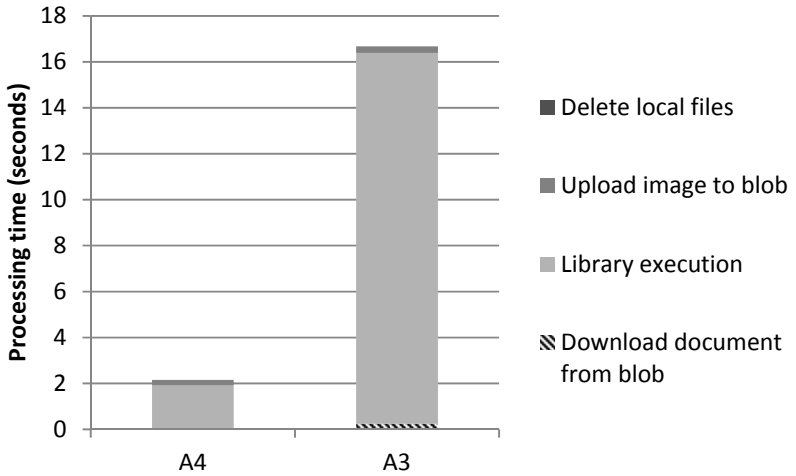

Figure 11 Page rendering time in the Cloud

As shown in Figure 11, pre-processing and post-processing steps for A3 page take about 0.5 seconds, which is only 3% of the total time (16.7 seconds). For A4 page these steps take 0.28 seconds, which is 13% of the total time (2.15 seconds). Cleaning local storage takes <1ms in both cases, so we can ignore this value. It is worth noting that downloading time is much shorter for A4 page, while uploading time is almost the same in both cases. This is because the difference in size between documents is much bigger than between output images. Anyway, library execution takes overwhelming majority of the time (>87%). This means CPU performance is still the most important factor for page rendering in the Cloud.

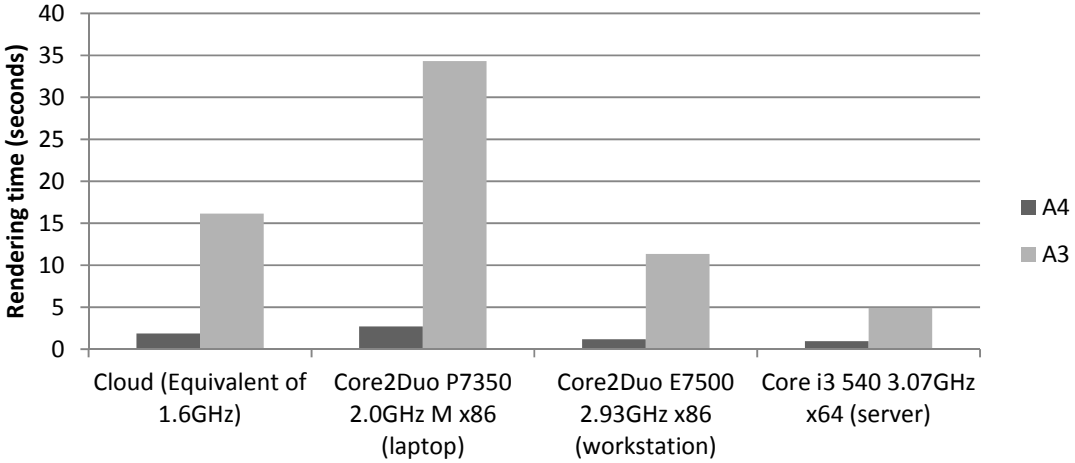

Figure 12 Page rendering time comparison for cloud and non-virtualized environment

As the next step, we compare page rendering time for cloud and on-premise DC versions. For a cloud version we use a small Azure compute instance (that has CPU performance equivalent to 1.6GHz), while on-premise installations have Core2Duo P7350 2.0GHz M x86 (laptop), Core2Duo E7500 2.93GHz x86 (workstation), Core i3 540 3.07GHz x64 (dedicated local server). Figure 12 illustrates the results of our experiments. We have observed notably worse performance of DC in the Cloud rather than on the dedicated server with powerful Intel Core i3 CPU (16.1 seconds compared to 4.9 seconds to render A3 page). This means the system needs about three times more instances of the backend engine in the Cloud to achieve the same throughput.

In order to verify that our results are reliable, we also run inferential statistics on the test data. We found that the standard deviation is <2% of the mean for all cases, due to a low variance for a big sample size. That makes **t** value to be very big (>100 for all cases), meaning that the probability of having an "error" is tiny (<0.0005). In our case the "error" implies that different test samples actually have the same mean while we have observed the opposite.

8.1.2. Session storing/retrieving time

In this section we compare on-premise and cloud DC session handling performance and also test two alternatives in Azure platform. For on-premise installation we evaluate standard ASP.NET in-process and state server modes. In-process mode stores session state data in memory, while state server mode uses a special process (separate from the ASP.NET worker process) for it. For cloud installation we evaluate AppFabric Cache, and a custom session handler that uses Azure Table.

Session handling is very important for the frontend ASP.NET application, because it retrieves and stores session data on every page load as a part of the ASP.NET application lifecycle [38].

After putting an object into session, we measure the time it takes to load and save the session when handling an http request. We perform this experiment against different storages and different object sizes: 1Kb, 1Mb, and 10Mb (assuming that session should not exceed 10Mb). Every object contains randomly generated binary data. It is worth noting that serialization time depends on the number of objects stored in session. In our case there is only one object. We also use the local Web server for state server mode, while a remote Web server would considerably increase session handling time. Experiment observations are presented in Table 3.

	On-premise DC installation		**Cloud DC installation**	
Session size	**In-process**	**State server**	**AppFabric Cache**	**Table Storage**
1Kb	0.0/0.0	0.0/0.0	0.004/0.008	0.094/0.113
1Mb	0.0/0.0	0.008/0.009	0.098/0.143	0.292/0.548
10Mb	0.0/0.0	0.161/0.173	0.435/0.583	1.167/1.861

Table 3 Storing/retrieving time for session data

As we can see in Table 3, on-premise DC requires significantly less time for session handling compared to the cloud installation. In-process mode is obviously the fastest. Since all data is kept in memory, the application needs to store and retrieve only a pointer to the memory location. However, when data is stored in another location like AppFabric Cache, it should also be serialized and de-serialized accordingly. We have observed that AppFabric Cache shows considerably better performance than Table Storage, especially for small amounts of data. It is approximately 3 times faster for 1Mb and 10Mb cases, and 17 times faster for 1Kb case (4/8ms compared to 94/113ms). Consequently, DC can have close to on-premise performance in the Cloud when operating smaller data amounts (kilobytes) stored in AppFabric Cache. Table Storage increases response time by $1.167+1.861 \approx 3$ seconds when storing 10Mb of data in session. On the other hand, it is much cheaper and has no capacity limits. Table Storage also shows a lower correlation between the time and session size. Apparently, this is caused by HTTP latencies to transfer the data.

8.1.3. Response time

In this section we test response time of the frontend web application against different deployment locations and different scale. We try to reflect the actual time from the end-user perspective, because perceived response time dictates user-friendliness of the service. For our case response time includes the latency to send a request from a client, the time to redirect the request by a load balancer (if there are several role instances), the time to process it by the application, and the latency to get a response back from the server (see Figure 13). Even though these factors depend on network locality and traffic congestion, the main purpose is to show the difference in response time depending on different conditions. In our experiments these variable conditions are deployment location, number of role instances (scale), and load. In order to measure response time purely for a Web Role, we use a stateless .aspx page that does not include any external factors like session handling or document page rendering.

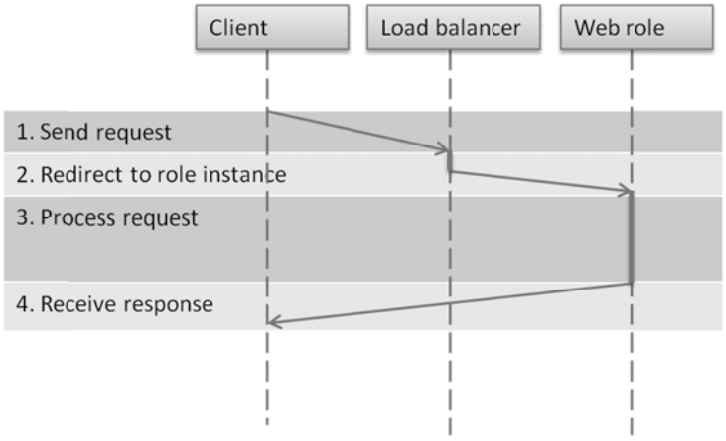

Figure 13 Page response time decomposition

The first experiment evaluates page response time for a different number of role instances. The page makes some calculations and then generates dynamic output content. This dynamic data is needed to ensure that the page is not cached by any CDN service or in the client environment. We perform the experiment with a variable number of simulated clients accessing the service concurrently. In order to measure response time, we use Visual Studio 2010 Load Test [39] based on a Web Test that simply requests the page. All testing is done from outside the Cloud. We run the Load Test for a period of five minutes and perform it many times to confirm that the results are stable.

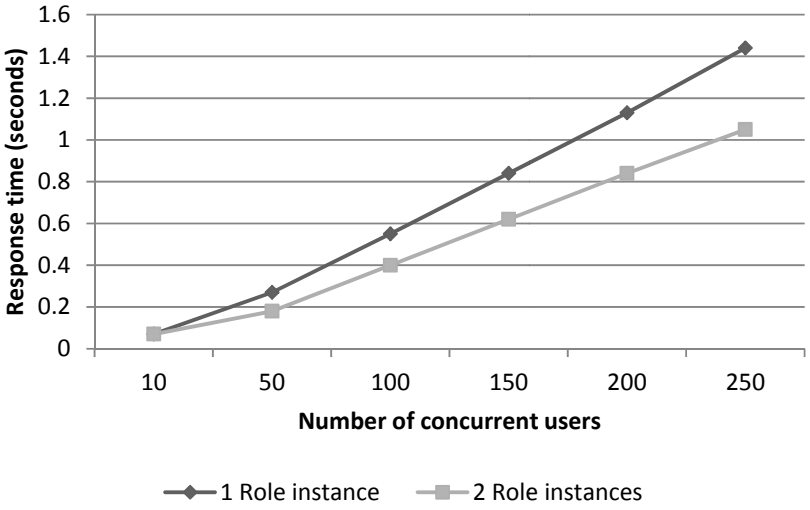

Figure 14 Cloud DC page response time

Figure 14 shows observed response time for both single instance and dual instance setups with an increasing number of concurrent users. Page time starts at about 80 ms for both cases and then glows linearly with different angular coefficients. Results show that an additional role instance decreases response time, especially for a heavy load. For 250 concurrent users a dual instance setup performs 400ms faster than a single instance setup.

The main goal of the second experiment is to show the difference in response time across different deployment locations. To do so, we execute Visual Studio 2010 Web Test that uploads a document to DC that is running in the cloud environment. This scenario reflects latency and bandwidth in a better way. We have picked up three random files of different sizes from the

production document set: 0.28Mb, 1.25Mb, and 5.13Mb. The experiment is executed for two deployment locations: North America and North Europe, with testing performed in Sweden (Gothenburg). We repeat the experiment multiple times to confirm that the results are stable. The observed response time is presented on Figure 15.

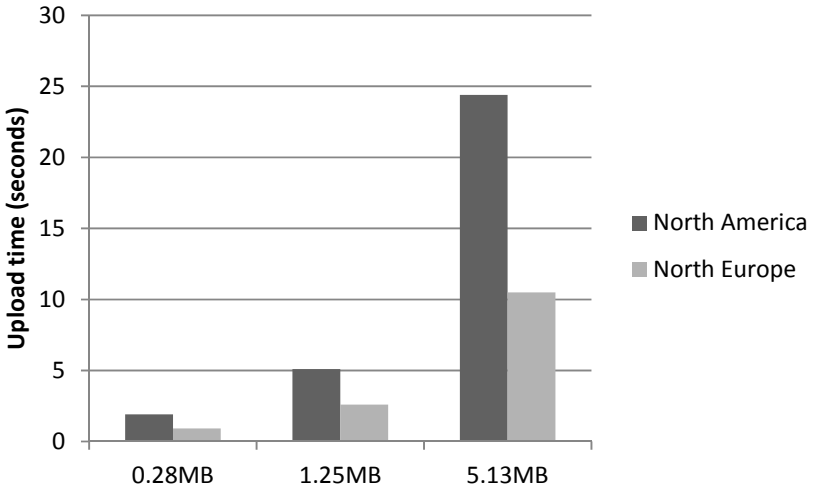

Figure 15 Response time for file uploading

All three cases show approximately twice faster uploading time for North Europe zone compared to North America zone. The biggest file (5.13 MB) is uploaded to the first zone for 10 seconds, while the second zone requires almost 24 seconds. Consequently, a proper deployment location can significantly improve user experience by reducing interaction latencies.

8.2. Cost

In this section we estimate the cost of DC in the Cloud. For this purpose we model three real life scenarios that describe how cloud DC can be used. The cost for every scenario is estimated based on the Microsoft Azure pricing model.

8.2.1. Scenario 1: demo installation

In Scenario 1 cloud-enabled DC is used as a demo installation. We have estimated the load and required capacities based on the statistics from current virtual private servers with demo installations. According to our estimations, we need three small compute instances: one for a Web Role and two for Worker Roles. The system needs a storage capacity of 100 GB and twice more (200 GB) for the outgoing traffic. We perform all calculations for a 30 days period which is equivalent to one month. So we totally need 30*24*3 = 2160 compute hours that costs 2160*0.12 = 259 US dollars. Data storage costs 100*0.15=15$; outgoing traffic – 200*0.15 = 30$; 1 million transactions cost only 1$; and 1 GB SQL Azure is 9.99$. The overall cost is presented in Table 4. Figure 16 shows the cost distribution among different services. The total cost in brackets represents an upfront payment case (using a subscription). For more information see the official Microsoft Azure page.

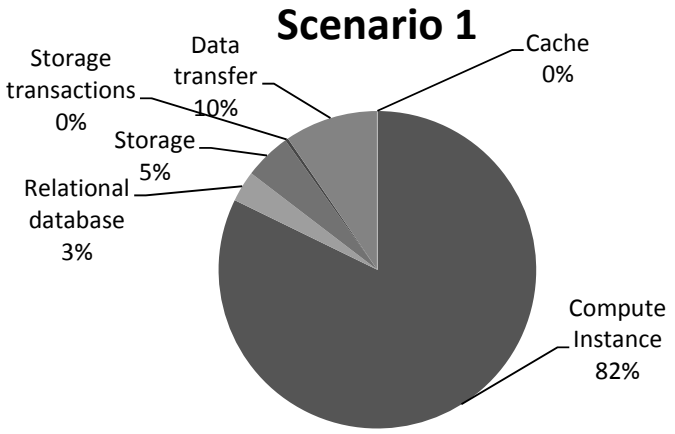

Figure 16 Cost distribution for Scenario 1

Service	Used capacity	Cost per month ($)
Compute Instance	3 small instances (2160 hours)	259
Relational database	1 GB	9.99
Storage	100 GB	15
Storage transactions	1000k transactions	1
Data transfer	200 GB	30
	Total:	314.99 (214.99)

Table 4 DC estimated cost for Scenario 1

8.2.2. Scenario 2: production installation without scaling

In Scenario 2 DC is used as a production installation with throughput equivalent to one dedicated server (without elastic scale). For this scenario we require DC to show the same throughout as the on-premise installation that is running on a server with Core i3 540 3.07GHz x64 processor, 500 GB available local storage and 4GB of memory. It uses three out of four cores for the Backend and the rest one for the Frontend. As we observed earlier, the backend engine shows three times worse performance in the Cloud. That means we need nine small compute instances for the Backend. The frontend application requires two small compute instances, since we do not expect big performance degradation for the ASP.NET application. We also include 512MB AppFabric Cache. The cost calculation is the same as for the previous case. Table 5 presents the total cost for this scenario, and Figure 17 illustrates the cost distribution.

Service	Used capacity	Cost per month ($)
Compute Instance	11 small instances (7920 hours)	950
Relational database	1 GB	9.99
Storage	500 GB	75
Storage transactions	5000k transactions	5
Data transfer	1000 GB	150
Cache	512 MB AppFabric Cache	75
	Total:	1264.99 (1084.99)

Table 5 DC estimated cost for Scenario 2

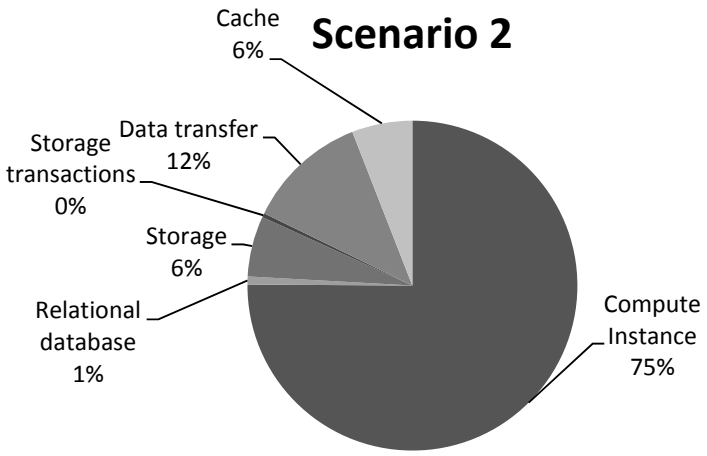

Figure 17 Cost distribution for Scenario 2

8.2.3. Scenario 3: production installation with scaling

In Scenario 3 DC is used as a production installation with throughput equivalent to one dedicated server (using elastic scale). In this scenario we use the same capacities as in Scenario 2, but leveraging cloud elastic scalability. We assume DC has a typical enterprise system load pattern: high load during working hours (10 hours from 8AM to 6 PM) and almost no load during the rest time. That means we can scale our system down when the load is very low. We scale it down to three small instances like in Scenario 1 to keep the system available. Also, the cache service is not needed when we have one Web Role. Assuming that there are 22 working days during a month we will need 30*24*3 + 22*10*8 = 3920 hours. The first term means that we need 3 instances all the time, and the second term means that we add 8 more instances during high load periods. The cache will cost 75*(22/30) = 55. However, using elasticity does not affect storage and outgoing traffic. The estimated cost is presented in Table 6. Figure 18 shows the cost distribution among different services.

Service	Used capacity	Cost per month ($)
Compute Instance	3-11 small instances (3920 hours)	470
Relational database	1 GB	9.99
Storage	500 GB	75
Storage transactions	5000k transactions	5
Data transfer	1000 GB	150
Cache	0-512 MB	55
	Total:	764.99 (664.99)

Table 6 DC estimated cost for Scenario 3

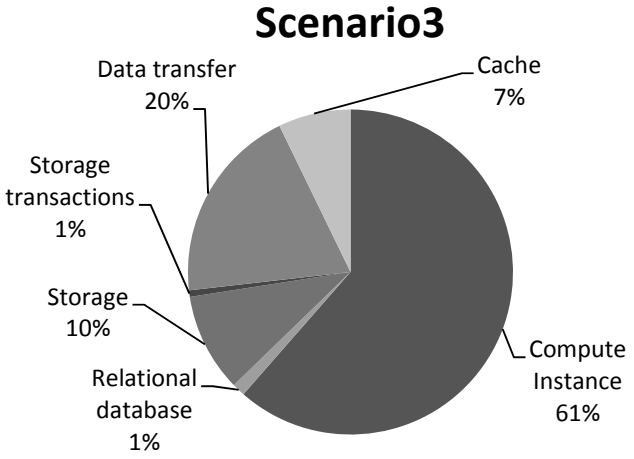

Figure 18 Cost distribution for Scenario 3

Based on our estimations we can conclude that compute services dominate in all scenarios. It makes up 82%, 75%, and 61% of the total cost for Scenario 1, 2, and 3 accordingly. On the other hand, storage transactions have the least cost. SQL Azure also has a small cost share: 1% for Scenario 2 and 3 and 3% for Scenario 1. However, this is because DC is not database centric. We found that the cost of DC can drop by 40 percent (764.99$ compared to 1264.99$) when leveraging elastic scalability. Even though choosing a proper scaling strategy is pretty straightforward for enterprise applications like ours, it might not be so trivial for other systems.

9. CONCLUSION

In this thesis we have taken an in-depth look at the current state of cloud computing. This technology is not mature yet, which brings difficulties in giving definitions and making classifications. Nevertheless, pay-as-you-go utility model, scalability and virtualization can be emphasized as the main cloud computing characteristics. Cloud computing is evolving and taking its shape rapidly, which is also reflected on cloud platforms dynamic. Thus, early adopters should be ready for a changeable weather in the Cloud.

Apart from cloud computing in general, we have made an extensive research on the opportunities and the challenges of cloud adoption. We have found that cloud computing enables a cost-efficient way of hosting highly available and geographically distributed applications that can de dynamically scaled based on the demand. On the other hand, cloud computing brings some challenges, including security, privacy, availability, and performance. We predict that many of these challenges will be solved or mitigated along with cloud computing maturing.

Furthermore, we have evaluated three leading IaaS and PaaS providers: Amazon AWS, Microsoft Azure, and Google AppEngine. Our findings support the argument that existing cloud implementations are idiosyncratic. We conclude that Amazon AWS is the most flexible platform that suites the widest range of applications, while Google AppEngine has the most limitations that are likely to complicate the migration. Microsoft Azure is an intermediate platform, particularly suitable for systems that are well-versed in Microsoft technologies.

Finally, we have implemented a cloud version of the on-premise enterprise application for Microsoft Azure platform. High DC compatibility with Azure and easy deployment were the main reasons for choosing this platform. The application cloud prototype was used to evaluate the performance and the cost of the system in a cloud environment. We have investigated the behavior of DC against different deployment locations, testing materials, scale and load. Our finding helped InformaIT to make a final decision regarding cloud adoption. Together with partners from InformaIT we have concluded that DC cloud implementation is feasible, despite degraded performance. We also found the estimated cost reasonable, especially when the system is dynamically scaled based on the load.

Extrapolation of the results

To our best knowledge there is no a unique metric that defines how well an application fits a cloud environment. The decision should be made separately for every system, based on the tradeoff between advantages and challenges. Existing systems are likely to face more challenges than new applications, due to the technological constraints of cloud platforms. In general, existing systems that are based on service oriented architecture with a focus on statelessness and low coupling fit the Cloud pretty well. Still, applications might require certain changes before being able to fully leverage a cloud environment. These changes are usually caused by environment limitations or the singularity of cloud storages.

Based on our observations, the cloud version of a system is likely to show worse performance because of higher latencies and inferior computing hardware underneath. In order to tune system performance, we suggest eliminating unnecessary transfers between different system components, meaning both the amount of data and the number of calls. In particular, web applications should reduce the amount of data stored in session or become completely stateless; data intensive applications should also consider using local cache to store frequently used data. HPC applications will usually require more CPU cores (compute instances) in the Cloud to show the same throughput. Thus, such applications are likely to be costly.

Last but not least, we suggest leveraging dynamic scalability in order to reduce the cost of a cloud application. This is especially important for systems with a changeable load. For example, enterprise application should scale up only during working hours; university web sites should scale up during application periods. However, monitoring is necessary when the load does not have a particular pattern. Furthermore, it might be ambiguous what metrics are the most relevant to monitor.

Future work

We have discussed many questions regarding the migration of applications to the Cloud. Still, there are some concerns of running applications in the cloud environment that we haven’t investigated. For example, we did not evaluate security, privacy, and availability of cloud-enabled services. Also, we did not test the scalability of cloud persistent storages.

Furthermore, it would be interesting to examine application behavior in other cloud platforms – like Amazon AWS – that also fit our case. Later, the results observed across different cloud environments could be analyzed and compared giving more comprehensive knowledge about the consequences of the migration.

We will also continue examining the DC system in the cloud environment in order to compare the estimated and the real costs. In addition to that, load and performance will be monitored so that we can suggest a better scaling strategy for DC.

REFERENCES

[1] D. F. Parkhill, “The Challenge of the Computer Utility”, *Addison-Wesley Educational Publishers Inc.*, US, 1966

[2] P. Botteri, D. Cowan, B. Deeter, A. Fisher, D. Garg, B. Goodman, J. Levine, G. Messiana, A. Sarin, and S. Tavel, “Bessemer’s Top 10 Laws of cloud computing and SaaS”, Bessemer Venture Partners, www.bvp.com, 2010

[3] N. Leavitt, “Is Cloud Computing Really Ready for Prime Time?”, *Computer*, vol.42, no.1, pp.15-20, Jan. 2009

[4] Andrew R Hichkey, “SMB cloud spending to approach $100 billion by 2014”, CRN, at http://www.crn.com/news/cloud/226700149/smb-cloud-spending-to-approach-100-billion-by-2014.htm, August 2010

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “A View of cloud computing”, *Communications of the ACM*, vol. 53, no. 4, pp. 50-58, Apr. 2010.

[6] M. Driver, “Cloud application infrastructure technologies need seven years to mature”. Research report G00162990, Gartner Inc., Stamford, USA, 2008

[7] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, “A break in the clouds: towards a cloud definition”, *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2009.

[8] Jeremy Geelan, “Twenty one experts define cloud computing”, *Virtualization*, Electronic Magazine, available at http://virtualization.sys-con.com/node/612375, Jan. 2010

[9] Microsoft Azure, http://www.microsoft.com/windowsazure/

[10] Google App Engine, http://code.google.com/appengine/

[11] Amazon Web Services, http://aws.amazon.com/

[12] Gordon Haff, “Just don’t call them private clouds”, CNET News, at http://news.cnet.com/8301-13556_3-10150841-61.html, Jan. 2009

[13] Andrew Conry Murray, “There is no such thing as a private cloud”, at http://www.informationweek.com/blog/229207922, InformationWeek, Jan. 2009

[14] L. Youseff, M. Butrico, and D. da Silva, “Toward a unified ontology of cloud computing”, *Grid Computing Environments Workshop, 2008. GCE '08* , pp.1-10, 12-16 Nov. 2008.

[15] GoGrid, http://www.gogrid.com/

[16] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslen, “Experimentation in Software Engineering: An Introduction”, *Kluwer Academic Publishers*, 2000

[17] J. W. Creswell, "Research Design", "Qualitative and Quantitative Approaches", *Sage*, 1994

[18] H.R. Motahari Nezhad, B. Stephenson, and S. Singhal, “Outsourcing Business to cloud computing Services: Opportunities and Challenges”, technical report, http://www.hpl.hp.com/techreports/2009/HPL-2009-23.pdf, 2009

[19] D. Chappell, “Windows Azure and ISVs: A guide for decision makers”, whitepaper at http://www.microsoft.com/windowsazure/Whitepapers/AzureAndISV/, Jul. 2009

[20] Wom Kim, Soo Dong Kim, Eunseok Lee, Sungyoung Lee, “Adoption issues for cloud computing”, Proceedings of the *11th International Conference on Information Integration and Web-based Applications & Services*, pp. 3-6, ACM, 2009.

[21] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, “Controlling data in the cloud: outsourcing computation without outsourcing control”, *Proceedings of the 2009 ACM workshop on Cloud computing security*, pp. 85-90, ACM, 2009.

[22] M.A. Vouk, “Cloud computing – Issues, research and implementations”, *30th International Conference on Information Technology Interfaces*, pp. 31-40, Jun. 2008

[23] B. Rimal, E. Choi, and I. Lumb. “A Taxonomy and Survey of cloud computing Systems”, in *INC, IMS and IDC, 2009. NCM’09. Fifth International Joint Conference on*, pages 44–51. IEEE, 2009.

[24] P. Louridas, “Up in the Air: Moving Your Applications to the Cloud”, *IEEE Computer Society Press Los Alamitos*, CA, USA, vol. 27, pp. 6-11, 2010.

[25] Ang Li , Xiaowei Yang , Srikanth Kandula , Ming Zhang, “CloudCmp: comparing public cloud providers”, *Proceedings of the 10th annual conference on Internet measurement*, November 01-30, 2010.

[26] Ang Li , Xiaowei Yang , Srikanth Kandula , Ming Zhang, “Comparing Public-Cloud Providers”, *IEEE Internet Computing*, vol. 15, pp. 50-53, Mar. 2011.

[27] Van Tran, Jacky Keung, Anna Liu, and Alan Fekete, “Application Migration to Cloud: A Taxonomy of Critical Factors”, *Proceeding of the 2nd international workshop on Software engineering for cloud computing*, pp. 22-28, ACM, 2011

[28] Muhammad Ali Babar, Muhammad Aufeef Chauhan, “A Tale of Migration to Cloud Computing for Sharing Experiences and Observations”, *Proceeding of the 2nd international workshop on Software engineering for cloud computing*, pp. 50-56, ACM, 2011

[29] B. Golden, “The Case Against cloud computing”, at http://www.cio.com/article/477473/, CIO, Jan. 2009

[30] R. Sean, “Cloud optimization – expanding capabilities, while aligning computing and business needs: A framework for making business decisions about cloud computing”, whitepaper at http://www.microsoft.com/windowsazure/Whitepapers/CloudOptimization/, Jun. 2010

[31] B. Glick, “IT leaders are ready for the cloud – but are their suppliers?” http://www.computerweekly.com/blogs/editors-blog/2011/02/it-leaders-are-ready-for-the-c.html, ComputerWeekly.com, 2011

[32] D. Durkee, “Why cloud computing Will Never Be Free”, *Queue*, vol. 8, pp. 20–29, ACM, 2010

[33] “Cloud computing survey: Exclusive research from CIO magazine”, http://www.cio.com/documents/whitepapers/CIOCloudComputingSurveyJune2009V3.pdf, CIO, Jun. 2009

[34] J. Hughes, “Encrypted Storage and Key Management for the cloud”, http://www.cryptoclarity.com/CryptoClarityLLC/Welcome/Entries/2009/7/23_Encrypted_Storage_and_Key_Management_for_the_cloud.html, 2009.

[35] Zach Hill, Jie Li, Ming Mao, Arkaitz Ruiz-Alvarez, and Marty Humphrey, “Early observations on the performance of Windows Azure”, *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pp. 367-376, ACM, 2010.

[36] “Shared Folders (SMB) Server”, http://technet.microsoft.com/en-us/library/cc734393 , 2011

[37] D. Templin, “Simplify App Deployment with ClickOnce and Registration-Free COM”, *MSDN Magazine*, at http://msdn.microsoft.com/en-us/magazine/cc188708.aspx, 2005

[38] “ASP.NET Application Life Cycle Overview for IIS 7.0”, at http://msdn.microsoft.com/en-us/library/bb470252.aspx, 2011

[39] “Running Web and Load Performance Tests”, at http://msdn.microsoft.com/en-us/library/ee923688.aspx, 2011

Appendix A: Microsoft Azure pricing model

Windows Azure

- Compute*
  - Extra small instance: $0.05 per hour
  - Small instance (default): $0.12 per hour
  - Medium instance: $0.24 per hour
  - Large instance: $0.48 per hour
  - Extra large instance: $0.96 per hour
- Virtual Network**
  - Windows Azure Connect - No charge during CTP
- Storage
  - $0.15 per GB stored per month
  - $0.01 per 10,000 storage transactions
- Content Delivery Network (CDN)
  - $0.15 per GB for data transfers from European and North American locations
  - $0.20 per GB for data transfers from other locations
  - $0.01 per 10,000 transactions

SQL Azure

- Web Edition
  - $9.99 per database up to 1GB per month
  - $49.95 per database up to 5GB per month
- Business Edition
  - $99.99 per database up to 10GB per month
  - $199.98 per database up to 20GB per month
  - $299.97 per database up to 30GB per month
  - $399.96 per database up to 40GB per month
  - $499.95 per database up to 50GB per month

Windows Azure AppFabric

- Access Control***
  - $1.99 per 100,000 transactions
- Service Bus
  - $3.99 per connection on a “pay-as-you-go” basis
  - Pack of 5 connections $9.95
  - Pack of 25 connections $49.75
  - Pack of 100 connections $199.00
  - Pack of 500 connections $995.00
- Caching
  - 128 MB cache for $45.00
  - 256 MB cache for $55.00
  - 512 MB cache for $75.00
  - 1 GB cache for $110.00
  - 2 GB cache for $180.00

    - 4 GB cache for $325.00

Data Transfers

- North America and Europe regions
    - $0.10 per GB in
    - $0.15 per GB out
- Asia Pacific Region
    - $0.10 per GB in
    - $0.20 per GB out
- Inbound data transfers during off-peak times through June 30, 2011 are at no charge.

*Compute hours are calculated based on the number of hours that your application is deployed.
**The Windows Azure Connect service is available in Community Technology Preview (CTP).
***No charge for billing periods before January 1, 2012.